

On the Use of Real-Time Agents in Distributed Video Analysis Systems

B. Lienard, A. Hubaux, C. Carincotte, X. Desurmont and B. Barrie

Multitel ASBL, Parc Initialis, rue Pierre et Marie Curie, 2, B-7000, Mons, Belgium

ABSTRACT

Today's technologies in video analysis use state of the art systems and formalisms like ontologies and datawarehousing to handle huge amount of data generated from low-level descriptors to high-level descriptors. In the IST CARETAKER project we develop a multi-dimensional database with distributed features to add a centric data view of the scene shared between all the sensors of a network.

We propose to enhance possibilities of this kind of system by delegating the intelligence to a lot of other entities, also known as "Agents" which are specialized little applications, able to walk across the network and work on dedicated sets of data related to their core domain. In other words, we can reduce, or enhance, the complexity of the analysis by adding or not feature specific agents, and processing is limited to the data concerned by the processing.

This article explains how to design and develop an agent oriented systems which can be used by a video analysis datawarehousing. We also describe how this methodology can distribute the intelligence over the system, and how the system can be extended to obtain a self reasoning architecture using cooperative agents. We will demonstrate this approach.

Keywords: Real-time, Video analysis, Agents, Middleware

1. INTRODUCTION

In today's video-analysis project, it is possible to extract features from video-scene and categorize the analyzed information to construct huge libraries of meta-data linked to the video. With search tools, it is possible to retrieve a part of a scene, at a certain time, in the whole scene based upon the research criteria given to the search tools. One example of this kind of service could be the security sector, where operators must often seek huge amount of video to track some details. Even if it seems very interesting, usual systems are based on a posteriori analysis which means that computers must analyze video during a long period and then track basic information. If more interpretation is required to offer more video search features, the analysis must be done in a couple of phases depending on the granularity (the level) of the feature. Another lack of these kinds of systems, is that it is not possible to directly query the system as long as the analysis are not performed. One solution is to plug directly, for instance in a video-surveillance context, the video sensor to a Video Content Analysis (VCA) and store the meta-data in a database for research.

Even if connecting a VCA to a database appears to be simple, it is not that easy in real-time systems. In fact, traditional database engines are not suitable for storage of huge amounts of meta-data available in real-time; indeed, it can be required that more complex algorithms need to look for low-level meta-data to perform their analysis of the scene. On the other side, it can be interesting for an operator to track an event identified 5 minutes earlier and return a couple of hours before this event without interrupting the current analysis.

In this article, we propose to explain how to build a system able to handle outputs of the Content Analysis and Retrieval Technologies to Apply Knowledge Extraction to massive Recording (CARETAKER) European IST project through a dedicated database architecture based on Agent Software Methodology. After a presentation of our system's concepts in Sec. 2, we will show which kind of video content analysis are applied to the video and audio stream and detail the system's context in Sec. 3. Finally we expose a concrete implementation of our agent based database in Sec. 4.

Further author information: Send correspondence to B. Lienard: E-mail: bruno.lienard@multitel.be, Telephone: +32 (0)65 34 27 53

2. DATAWAREHOUSING AND AGENT

In a previous paper,¹ we already introduced the usage of a data-mining system called Datawarehouse, in a network of video sensors, to enhance the analysis capability of sensors. Instead of trying to communicate information between all sensors, we may have a data centric view of all the information available on the network where each sensor updates a global model of a complete scene related to its context. However, bases of this system must be improved to fit a more massive network where both audio and video are stored on dedicated databases and where external retrieval must be done with minimum latency. In fact, one of the feature of the CARETAKER system, is to orchestrate a system which can connect low-level video-analysis stub (algorithms responsible of basic feature extraction) with the high-level video-analysis stub (algorithms responsible of context and scenario feature extraction) and thus provide a powerful, real-time base, meta-data storage and retrieval system. In other words, low, middle and high level meta-data must be handled by a single system which are than provided to both “actors” in the system (a high-level stub can require ”in the past” low-level information as well as a human operator can ask for the tracking of a recent high-level event).

A major innovation of CARETAKER in surveillance systems is to apply joint audio and video content analysis, knowledge modelling and extraction techniques in real-use cases with real data produced by a network of AV sensors. The project proposes a hierarchical modelling of the semantic content: a first layer of primitive events, extracted from the raw data streams and a second layer of higher semantic events from which rich meta-data can be produced. The semantic description will be used in two systems: a web service-oriented on-line prototype, demonstrating the validity of the primitive and some mid-level events extractors (the design of the on-line subsystem is illustrated in Fig. 1); which will serve as a demonstrator applying the technology in a real-time system handling from 10 to 20 video cameras and microphones. The purpose of this small-scale, on-line, prototype system is to demonstrate both the effectiveness of the technology, and how the implementation can scale to the massive recording scenarios of real-scale monitoring systems. To enable the results of low-level stubs (quoted as A in the schema) and high-level stubs (quoted as D in the schema) to be used in a real-scale environment with an abundance of sensors, state of the art open standards for distributed event driven processing will be used. In particular, web-service technology (SOAP, WSDL, UDDI, RSS) is chosen for components and subsystems integration, because it allows reuse of high-performance interoperable components and makes the required distributed processing and communication more straightforward. Key components of this task are the analysis of system scalability and how web service technology can be deployed in order to achieve effective semantic knowledge acquisition on a massive scale.

The low-level semantic subsystems, or “clusters”, process sensor signals (A) from one or more cameras. The sensor data is stored in a dedicated audio video store (B) and the low-level semantic descriptors resulting from the analysis are stored in a local relational database (C). This low-level semantic data recorded in the relational database is the output from low-level generator stubs technology, i.e. features and tracks captured from one or more sensors observing a common scene. Between the subsystems, query and retrieval of both descriptive and representative data semantic data will use web-service standards (e.g. RSS feeds for newly detected events).

At the core of the high-level semantic layer is the data-warehouse (E): a multidimensional database (e.g. using OLAP) used for data-mining and semantic inferences. It is populated by several independent processes (D), that retrieve data from the low level clusters. These processes store in the data-warehouse all low-level descriptors, create and store new high-level descriptors. The rules for constructing the high-level descriptors (notably responsible for alerting the operator to events of interest) result from the high-level stubs (D) work.

The interface (F) with the operator (user) is mediated by a Web Server (or any other Web-Service client) with the capability of dynamic pages publication (e.g. JSP and PHP). The Web Server is also able to format queries output from the data-warehouse into human readable form. This interface will also mediate the request and retrieval of original audio and video data for specific events of interest. A Graphical User Interface will be integrated for the configuration mode, the queries and the feedbacks during in the training phase. Feedback about the relevance of events can also be used in the operational mode to on-line adaptation of the detection and sensitivity rates.

Thus to fit to requirements of the prototype we construct the datawarehouse using state of the art technologies of the Knowledge Management. The main idea is to optimize a Resource Description Framework (RDF)

serialization of meta-data and store it in an optimized way in database. So since we must reduce latency of the system to obtain a real-time system we also focus on the multi-threading methodology and specially on the Agent Oriented Programming, where piece of software are responsible of formatting and organization of data in the datawarehouse. Other agents are also developed to infer on data to speed up queries and link, a priori, distinct datasets.

One other key feature is that our system must be open and generic, this means that it is not aware of the kind data it must handle before its initialization. Indeed, we delegate the responsibility of the data definition to meta-data generator stubs, using a standard schema description language syntax. This syntax is the simplified version of the Relax NG grammar which is interpreted at run-time to construct data-structures. However, once the system is initialized with schemas coming from both levels stubs, the schemas used must not be altered during the life-time of the database (it is not yet possible to change the data structure on the fly).

Finally we would like to create an auto-configurable system which is able to publish low-level meta-data from the low-level stubs generators through a web-service like mechanism a.k.a. RSS Feeds. I will also be possible to structure and integrate data in the data warehouse using a RDF representation, to forward low-level meta to middle and high-level generators and to integrate outputs of these stubs in the data warehouse. On the other side of the system, a Web-Service like gateway must be also available to query the datawarehouse by means of an operator. All internal processing is done in an asynchronous way by autonomous piece of software called Agent.

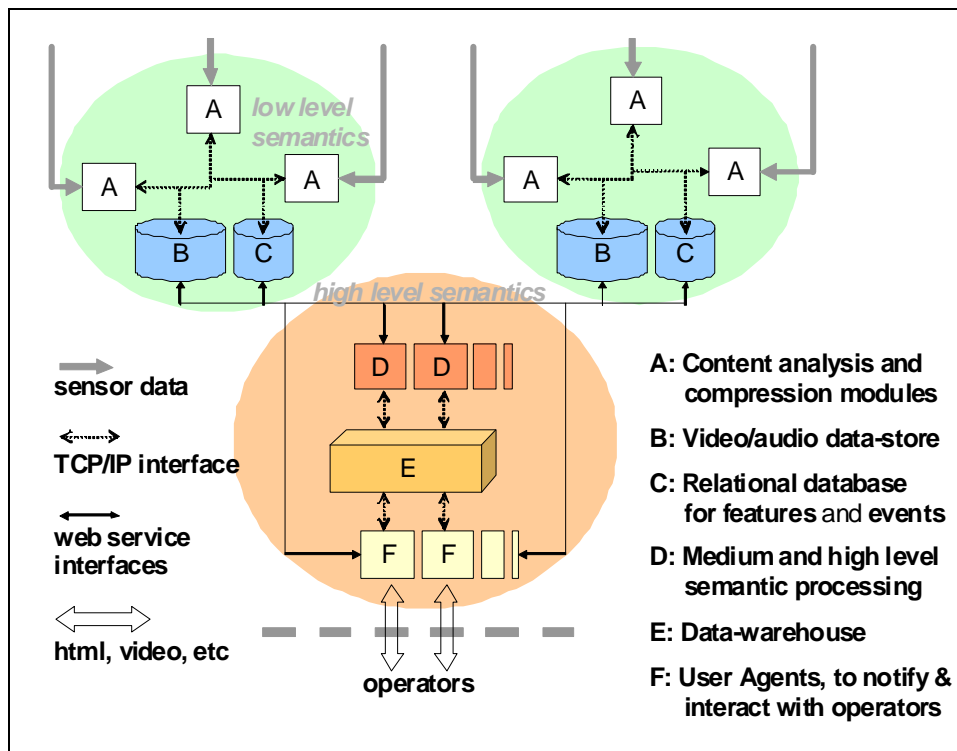


Figure 1. Diagram to illustrate the likely design configuration for the real-time subsystem.

3. VIDEO CONTENT ANALYSIS TECHNIQUES

Advances in sensor devices, communication and storage capacities make it increasingly easier to collect large amount of video material. However, the value of this recorded data is only unlocked by technologies that can effectively exploit the knowledge it contains. It is thus the goal of the proposed distributed analysis system to demonstrate the effectiveness of real-time agents in a video environment.

In this context, we will focus on video content analysis techniques allowing the automatic extraction of semantic metadata from raw multimedia. The motivation is that video networks are becoming more and more common in different environments such as public transportation premises, cities, public buildings or commercial establishments; the multimedia streams produced could thus potentially represent a useful source of information if stored and automatically analyzed, for instance in urban planning and resource optimization applications.

Relevant content analysis within video data classically requires low-level vision processing of the image and of the moving objects it contains. Indeed, a low-level representation of the objects in the scene is mandatory to perform a reliable semantic analysis. In a second stage, inference processes operate at the representation level to identify more complex semantic contents. In this context, we consider two types of content knowledge: a first layer of primitive features that can be extracted from the raw data streams, such as presence of “object” in the scene, description of object, paths followed by object, . . . A second layer of higher semantic events is defined from longer term analysis and from more complex relationships between the primitive features. Next two sections are respectively devoted to these two level of knowledge.

3.1. Low-level content extraction

In video-surveillance and monitoring applications, trajectories of object/people in the scene are some of the most obvious content to extract. As a matter of fact, background segmentation is usually the first step of the image process chain. A common method for real-time segmentation of moving regions in image sequences has been proposed by Stauffer and Grimson;² this technique models each pixel as a mixture of Gaussians and uses an on-line approximation to update the model. Fig. 2 presents an example of moving object detection obtained with this approach.



Figure 2. Moving object detection results obtained through Gaussian Mixture Model.

Once the background/foreground estimation is achieved, in order to reduce the amount of information to what is strictly necessary for high level inference, a description process “translates” video data into a symbolic representation (i.e. descriptors). For each image at time t , the description process computes k different features for each observed blob (object) in the corresponding foreground map. More precisely, these features are $2D$ position in image, $3D$ position in the scene, bounding box, mean RGB colour, $2D$ visual surface, inertial axis of blob shape, extreme points of the shape, probability to be a phantom blob, . . . Fig. 3 presents some results obtained through this description process; mean RGB colour (b), ellipse (b) and bounding box (c) allow to provide the high level inference with relevant information.

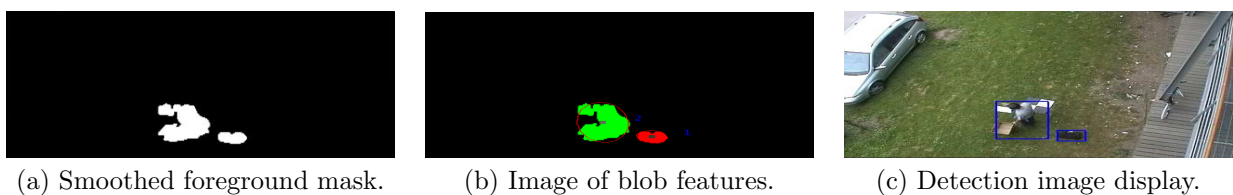


Figure 3. Description results obtained through the process described above.

Tracking algorithms then allow to follow moving objects of the scene along time. In our case, we use a bottom-up tracking with multiple matching decisions (multiple hypothesis tracking³). This section does not

*Image composed of the means of the most probable Gaussians in the background model.

intend to give the detailed description of the tracking process used; interested readers may consult Desurmont *et al* chapter.⁴ Nevertheless, the tracking process is divided into four main steps that follow a straightforward approach: Prediction, Cost matrix, Matching decision(s), and Tracks update(s):

- Prediction is quite basic. It predicts the blobs features (position, colour, size,...) through a recursive estimator in a *Maximum A Posteriori* framework.
- Cost matrix computation aims at identifying a cost $c_{i,j}$ for a matching between blobs i and j respectively in current and previous frame.
- Matching is achieved according to the computed cost matrix thanks to several matching algorithms (one algorithm at a time).
- Tracking update provide some complementary information such as time of life of every blob of a track, time before “death”,... Tracking filtering is also performed on the tracking description output.

Fig. 4 presents some tracking results obtained through this bottom-up approach. Fig. 4-(a) and (b) show how the tracking updates can be useful to clean the tracks. Note that A stands for the *apparition of new target*, S for *split of track* and M for *merge of tracks*.

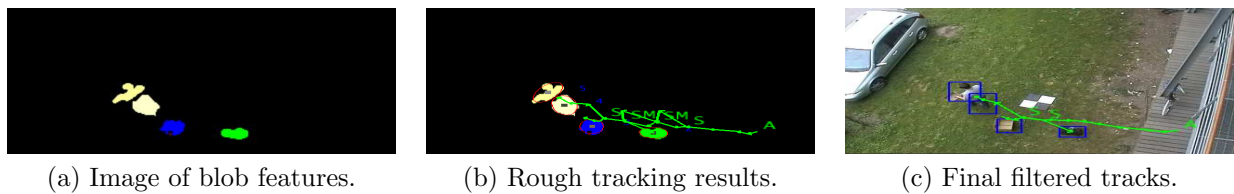


Figure 4. Tracking results obtained through the bottom-up approach described above.

3.2. High-level content analysis

The high-level content analysis is a process that receives the tracking description. In addition to tracking description results, the high-level analysis also requires contextual information, which is usually represented by means of 2D polygons on the image, each of them having a list of attributes: *IN/OUT zone*, *NOISY zone*, *OCCLUSION zone*, *AREA OF INTEREST zone*,... This type of information enables the improvement of the scenario recognition and of the alarms management processes (e.g. to ring the alarm when somebody tries to enter through an exit door). The high-level analysis can then find predefined patterns like objects entering in a defined zone of the image and exiting by another one, objects which have exceeded a certain speed limit, or object(s) immobile from a certain time stemming from another mobile object (abandoned luggage),...

Fig. 5 shows a particular pattern of high-level event detection, namely the abandoned luggage detection. This high-level event can be identified through a defined scenario: *apparition of a mobile target* (a), *immobile object stemming from the mobile target* (b) and *immobile objects exceeds a predefined maximum time* (c).

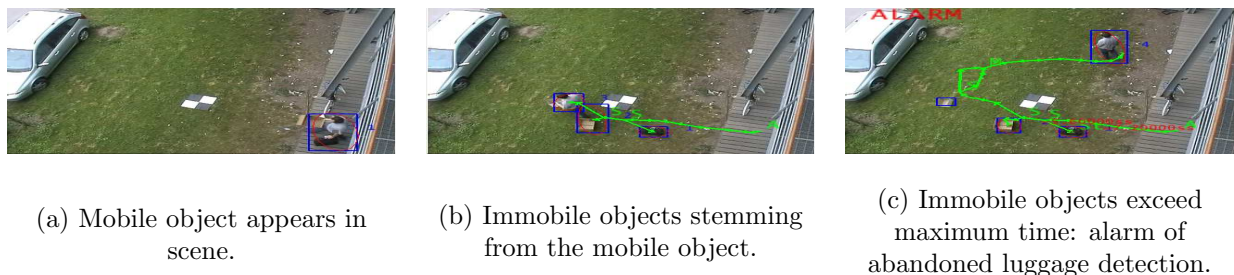


Figure 5. High-level content analysis results (abandoned luggage scenario).

4. AGENT-BASED DATA WAREHOUSE

To describe the system to build we will first define how data is exchanged among entities. Then, we will focus on the architecture of the agents system. The last two points will be dedicated to the structure of the data stored and the data warehouse.

4.1. Data exchange

In order to define the way data is exchanged, the first point was to specify the type of transferred data. The system will have to process three kinds of XML formatted data: results from analysis, queries and queries' results. To avoid sending raw files over the network, every transferred information is wrapped in an RSS feed. The data exchange was thus reduced to the handling of an RSS flow. Unlike common RSS news readers polling registered sites at regular times to fetch new events as showed in Fig. 6(a), we favoured a *publisher/subscriber* approach⁵ described in Fig. 6(b). Indeed, in a real time environment the waste of time entailed by the polls and the useless consumption of resources are unacceptable. Moreover, the use of RSS enables the addition of complementary fields to the raw XML, e.g. content identifiers and the source file used by the high-level analysis.

To restrict the overhead appended to the original file, we do not resort to any high level protocol and directly send RSS feeds over a TCP/IP connection.

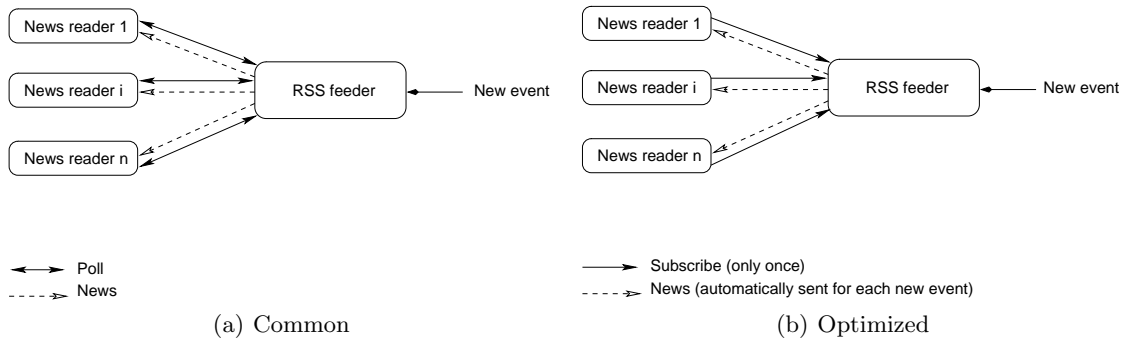


Figure 6. RSS architectures

4.2. Agents system

With the intention of building a highly reactive system, the agent paradigm turned out to be the most suitable choice. Indeed, agents offer a high degree of flexibility and autonomy and are much more reactive than sequential systems.

The system is composed of two kinds of agents: *process-oriented* and *data-oriented* agents.

Process-oriented agents were implemented with JACK,⁶ a Java based agent-oriented language and JDE, the JACK Development Environment.⁷ These agents are in charge of the RSS and data flow handling. Fig. 7 sketches the interactions between them.

Data-oriented agents, implemented in pure Java, are named *autotroph agents*. If you refer to biology, autotrophs are organisms able to create organic compounds, from inorganic compounds. As a result, they are always the basis of any food chain. Autotrophic replicators are thus able to reproduce themselves by building their own resources from substances whose nature is not equivalent to the ones constituting them.

The data-oriented agents we use are close to these organisms. Indeed each agent is capable of self-replication, self-reification, self-export to XML, RDF/XML, N-Triples, performing optimizations on the structure of the produced RDF graph, updating their elements' values, modifying their internal structure from raw data and merging with *autotroph agents* of different types. Every process in the system handles these agents. They can be seen as the smallest manageable elements.

The structures, also called skeletons, of these agents are obtained from *Relax NG* files defining the schemas of

the various XML files wrapped in the RSS feeds. When a new XML file is received, its type is analysed and the matching skeleton is replicated. The structure of the replicated agent is then adapted and instantiated with the values enclosed in the XML file.

The internal, multiply chained, structure of the agent is depicted in Fig. 8[†]. Each element of the structure can be seen as a directly addressable sub-agent capable of the operations described above, which enables an awesome handling flexibility.

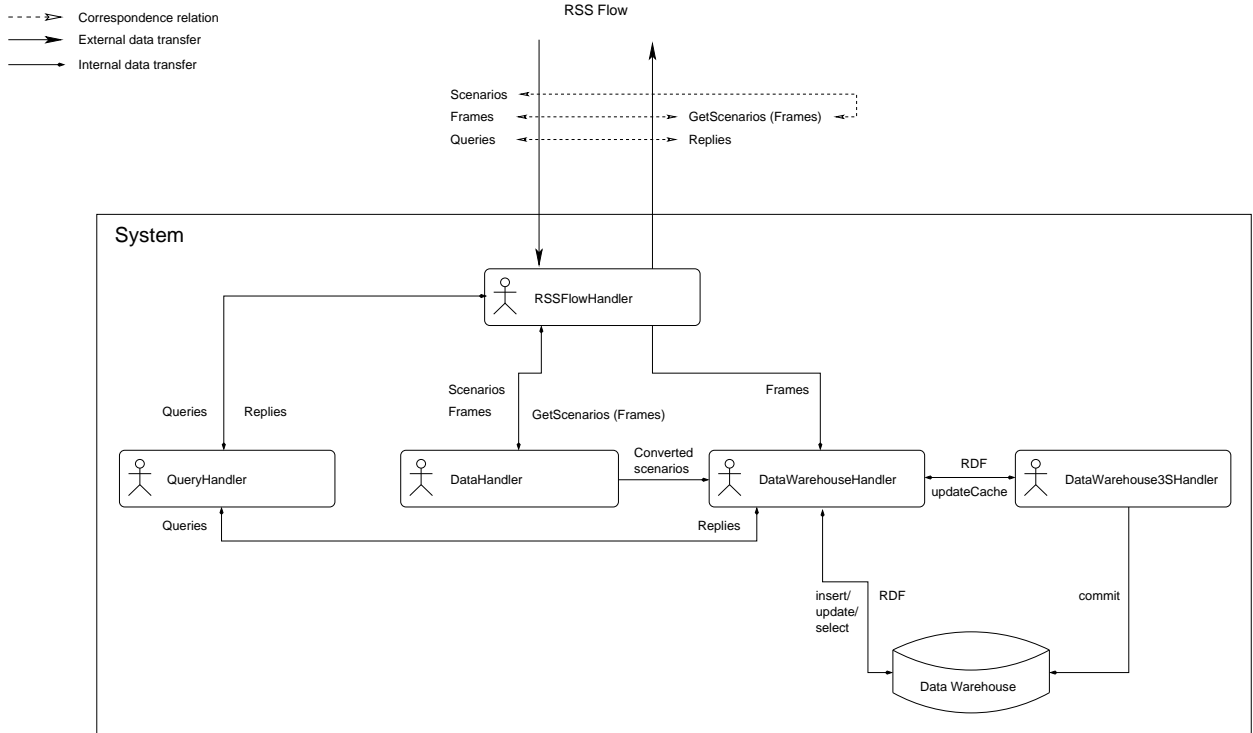


Figure 7. System's architecture

4.3. Stored data structure

In a real time environment, data storage is one of the most intricate issues. The amount of data is huge, the throughput is very high, the queries must be handled as fast as possible, the analysis' results contain lots of duplicated elements and the schemas used are not *a priori* known, ruling out any *a priori* optimization. Moreover, the *translation* of the XML tree structure into a relational database could entail a fairly high number of referential constraints. We thus decided to resort to RDF, a technology commonly used in the semantic web.

RDF graphs are sets of triples, also called *statements*, of the form (*subject predicate object*). The predicate is a directed arc going from the subject to the object. It can be seen as a property linking the object to its subject. More formally,⁸⁻¹¹ if we define U as an infinite set of RDF URI references, $B = \{N_i : i \in \mathbb{N}\}$ as a set of RDF blank nodes and L as an infinite set of RDF literals (e.g. string, int, ...). A triple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ where s is a subject, p a predicate and o an object is called an RDF triple and a graph $G \subseteq (U \cup B) \times U \times (U \cup B \cup L)$ is an RDF graph. Note that strictly speaking, RDF are rather ordered hypergraphs than graphs.¹²

The main advantage of RDF in our case is that duplicated literals in XML files will not be duplicated in the data warehouse. Instead, a new predicate will be added between the newly inserted node and the previously saved literal. Unfortunately, this graph-based approach doesn't prevent deep tree structures. To avoid following

[†]Implementation-specific variables were omitted to avoid overloading the figure with secondary information.

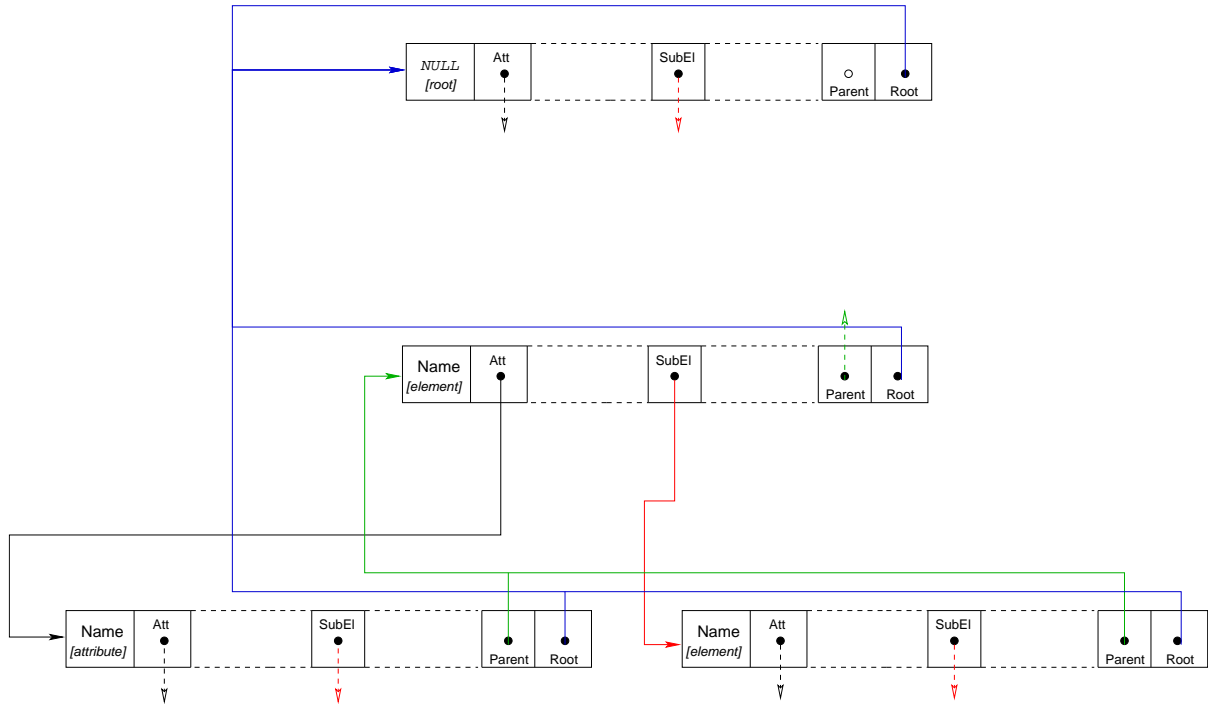


Figure 8. Autotroph agent's structure

long paths occurring in XML files, some optimizations on the structure of the schemas had to be performed, which is one of the tasks of the autotroph agents. In order to speed up reification time and to reduce the space required to store subgraphs, every XML file exported by the autotroph agent as an RDF has a depth of 4 nodes, no matter the schema of the XML file. The generic RDF Schema (RDFS) corresponding to this graph is the following:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="urn:xml2rdf/#">

  <!-- CLASS definitions -->
  <rdfs:Class rdf:ID="relaxNG_Type_Class"/>
  <rdfs:Class rdf:ID="instance_ID_Class"/>
  <rdfs:Class rdf:ID="field_ID_Class"/>
  <rdfs:Class rdf:ID="field_type_Class"/>

  <!-- PROPERTY definitions -->
  <rdf:Property rdf:ID="hasInstance" >
    <rdfs:domain rdf:resource="#relaxNG_Type_Class"/>
    <rdfs:range rdf:resource="#instance_ID_Class"/>
  </rdf:Property>
  <rdf:Property rdf:ID="instContent" >
    <rdfs:domain rdf:resource="#instance_ID_Class"/>
    <rdfs:range rdf:resource="#field_ID_Class"/>
  </rdf:Property>
  <rdf:Property rdf:ID="hasContent" >
    <rdfs:domain rdf:resource="#instance_ID_Class"/>
    <rdfs:range rdf:resource="rdfs:Literal"/>
  </rdf:Property>
```

```

<rdf:Property rdf:ID="typeInstance" >
  <rdfs:domain rdf:resource="#field_type_Class"/>
  <rdfs:range rdf:resource="#field_ID_Class"/>
</rdf:Property>
</rdf:RDF>

```

The `instance_ID_Class` class is used to restrain values returned by queries to subjects with the matching type.

This enables the reification of each XML file's corresponding agent in a single SPARQL query of the form:

```

SELECT ?fieldID ?hasContent ?content
WHERE { <instance_ID> <instContent> ?fieldID . ?fieldID ?hasContent ?content . }

```

Unlike semantic web applications carrying out inferences based on RDFS and OWL (Web Ontology Language) definitions, the system does not perform any of them. The reason is simply that they are too time and resource consuming.

4.4. Data warehouse

The use of RDF to store data implies the choice of a storage technology. Many attempts have been made to find the most suitable way of organising data.¹³⁻¹⁷ Some use RDFS and/or OWL where others use a main table containing triples. Graphs may be stored in an object-oriented database,¹⁸ in relational database or even in text files. Our main goals were to store huge amounts of data and to minimize the steps separating the extraction of the XML file from its RSS feed and the *physical* storage of the corresponding RDF.

We choose *3store*¹⁴ a MySQL backed RDF triple store implemented in C. The main advantages of *3store* are that it performs a reduced set of inferences, it benefits from optimizations realized in MySQL, it fully supports the SPARQL query language and it provides a very simple and efficient interface. The two main tables of the database are `symbols` and `triples`. The `symbols` table contains node' values and their corresponding hashed value. The `triples` table contains the triples where node' values are replaced by their corresponding hash. This table also contains a fourth column named `model` allowing the definition of graphs contexts. So *3store* is more a *quad store* than a *triple store*. This schema enables faster queries and speeds up tests on duplicated nodes.

To improve the system's response time to queries, we implemented our own cache using a red/black tree. A red/black tree is a self-balancing binary search tree guaranteeing queries, insertion and deletions in $O(\log n)$ time. Each node of the tree contains one autotroph agent. The access key of each node is the creation ID of the associated agent.

Given the size and incoming rates of data, the cache size has to be limited to avoid excessive swapping and main memory overflow. In order to meet the imposed real time constraints, the size must be determined by the needs of the high-level analysis. The cache size has thus to be as close as possible to the maximum interval between the latest received file, triggering a high-level analysis, and the oldest previously received file used by the triggered analysis. So, if we assume that the oldest file used is 5 minutes old and that the system receives 25 files per second, the number of agents to keep in cache would be $5 \times 60 \times 25 = 7500$, i.e. a 7500-node red/black tree.

Fig. 9 sketches the context of the data warehouse. The targets of each agent are identified and the way they interact with each data components is illustrated.

4.5. Performance evaluation

Tests were performed with sample XML files retrieved from video sequences. Simulators were used to send low/mid/high-level analysis and queries. The following table lists the results of some of the most recurring operations. The XML test file used here comprised 53 instantiated values. The *3store* knowledge base contained more than 4 000 000 triples, corresponding to approximately 13 minutes of recording at 25 files per second.

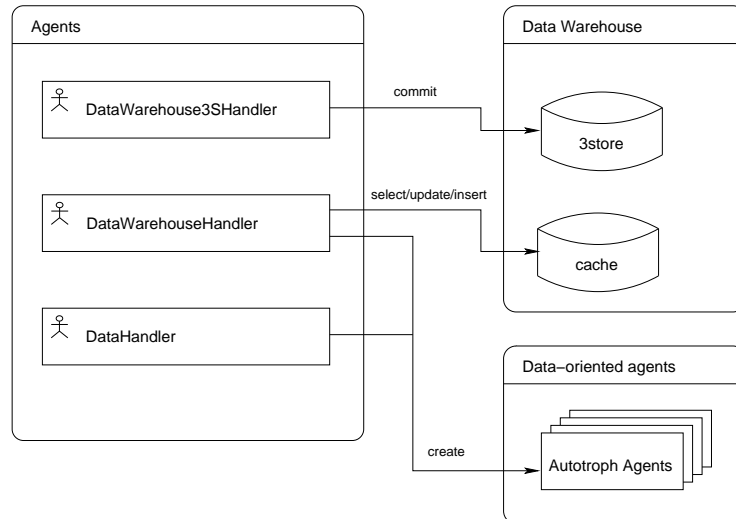


Figure 9. Data warehouse context

Operation	Average time (ms)
Reification (from cache)	0
Reification (from 3store)	40
Query (from cache)	6
Query (from 3store)	50
Agent replication	0
Agent initialization	5
Agent → XML	5
Agent → RDF	5
Insertion (in cache)	0
Insertion (in 3store)	70

More precise tests must be conducted but the current levels of development of the analysers do not allow real scale tests. These measures must thus be regarded as indicative.

5. CONCLUSION AND PERSPECTIVES

The system we have built makes use of intelligent agents to process data in a real-time environment. We have also developed special kinds of agents called *autotroph agents*. The operations they can perform improve and speed up the way data are handled. RDF graphs were used to efficiently store the huge amount of data received and to promptly answer queries. The physical distribution of processing units guarantees a very flexible and highly capable system. Furthermore, local tests carried out showed very good results regarding insertion and query times.

We are currently developing features specific masks that will be added to the *autotroph agents* to restrict the amount of data processed. Moreover, the system's scalability enables the insertion of specialized agents such as VCA capable agents. Future works will also include real scale evaluations.

ACKNOWLEDGMENTS

The work presented here is partially supported by the European Commission under the 6th Framework Program through the CARETAKER project (Activity: Semantic-based Knowledge and Content Systems, contract no.: FP6-027231). For further information about the CARETAKER project, please visit <http://www.ist-caretaker.org/>.

REFERENCES

1. B. Lienard, X. Desurmont, B. Barrie, and J.-F. Delaigle, "Real-time high-level video understanding using data warehouse," in *Real-time Image Processing III*, January 16-19 2006.
2. C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 246–252, 1999.
3. I. Cox and S. Hingorani, "An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**, pp. 138–150, Feb. 1996.
4. X. Desurmont, A. Bastide, J. Czyz, C. Parisot, J.-F. Delaigle, and B. Macq, *Intelligent Distributed Video Surveillance Systems*, ch. A General-Purpose System for Distributed Surveillance and Communication. S.A Velastin & P Remagnino Eds, Institution of Electrical Engineers, London, 2005.
5. M. Petrovic, H. Liu, and H.-A. Jacobsen, "G-topss: fast filtering of graph-based metadata," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pp. 539–547, ACM Press, (New York, NY, USA), 2005.
6. Agent Oriented Software Pty. Ltd., *JACK Intelligent Agents: Agent Manual*, 2006.
7. Agent Oriented Software Pty. Ltd., *JACK Intelligent Agents: Development Environment Manual*, 2006.
8. F. Frasnjar, G.-J. Houben, R. Vdovjak, and P. Barna, "Ral: An algebra for querying rdf," *World Wide Web* **7**(1), pp. 83–109, 2004.
9. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle, "The ics-FORTH RDFSuite: Managing voluminous RDF description bases," in *SemWeb*, 2001.
10. G. Karvounarakis, A. Magganaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle, "Querying the semantic web with rql," *Comput. Networks* **42**(5), pp. 617–640, 2003.
11. C. Gutierrez, C. Hurtado, and A. O. Mendelzon, "Foundations of semantic web databases," in *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 95–106, ACM Press, (New York, NY, USA), 2004.
12. R. Angles, C. Gutierrez, A. Gomez-Perez, and J. Euzenat, "Querying rdf data from a graph database perspective," *Lecture notes in computer science*, 2005.
13. A. Owens, "Semantic storage: Overview and assessment," 2005.
14. S. Harris and N. Gibbins, "3store: Efficient bulk rdf storage," 2003.
15. Y. Theoharis, V. Christophides, and G. Karvounarakis, "Benchmarking database representations of rdf/s stores.," in *International Semantic Web Conference*, pp. 685–701, 2005.
16. B. Liu and B. Hu, "An evaluation of rdf storage systems for large data applications," *skg* **0**, p. 59, 2005.
17. L. Ding, K. Wilkinson, C. Sayers, and H. Kuno, "Application-specific schema design for storing large rdf datasets," 2003.
18. V. Bonstrom, A. Hinze, and H. Schweppe, "Storing rdf as a graph," in *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, p. 27, IEEE Computer Society, (Washington, DC, USA), 2003.